MISB MOTION IMAGERY STANDARDS BOARD	
	MISB ST 1901.1
STANDARD	
Motion Imagery Metadata (MIMD): Modeling Rules	**25 June 2020**

1 Scope

The Motion Imagery Metadata (MIMD) documents are composed of the MIMD Modeling Rules, MIMD Model, and Model-to-KLV Transmutation Instructions. This standard defines the MIMD Modeling Rules. The modeling rules are a subset of standardized UML class diagrams [1] specifically minimized to enable transforming, or transmuting, class instances to KLV. Other formats such as XML or JSON may have transmutation instructions developed for them in future revisions. MISB ST 1903 [2] defines the MIMD Model, which is a UML model organized as a hierarchy of information including metadata for temporal, platform, payload, sensor, command, automated processes, exploitation, security, and more. MISB ST 1902 [3] defines the Model-to-KLV Transmutation Instructions for constructing bandwidth efficient KLV structures from the model. This document, in concert with MISB ST 1902, MISB ST 1903 and its supporting MIMD model detail documents, provides a suite of standards embodying the MIMD architecture.

2 References

[1] OMG® UML.2.5 Unified Modeling Language®, 1 December 2017.

[2] MISB ST 1903.1 Motion Imagery Metadata (MIMD): Model, Jun 2020.

[3] MISB ST 1902.1 Motion Imagery Metadata (MIMD): Model-to-KLV Instructions Transmutation Instructions, Jun 2020.

[4] MISB MISP-2020.1: Motion Imagery Handbook, Oct 2019.

[5] MISB ST 0107.4 KLV Metadata in Motion Imagery, Feb 2019.

[6] MISB ST 1904.1 Motion Imagery Metadata (MIMD): Base Attributes, Jun 2020.

[7] ISO/IEC 10646:2014 Information technology – Universal Coded Character Set (UCS).

[8] IEEE 754-2008 Standard for Floating-Point Arithmetic [and Floating-Point formats].

[9] MISB ST 1201.4 Floating Point to Integer Mapping, Feb 2019.

[10] BIPM The International System of Units (SI) 9th Edition, May 2019.

3 Revision History

Revision	Date	Summary of Changes
ST 1901.1	6/25/2020	• Clarified naming rules and equivalence for attribute names • Added Boolean type • Added Boolean Array type • Added IntegerId type for the mimdIds • Added RESERVED as an attribute type (placeholder) • Clarified that all integer and UInt types use a resolution of one • Changed ‘List’ to ‘LIST’ for all lists • Changed ‘*classname*_Ref’ to ‘REF<*classname*>’ • Added MIMD Model Document Structure which supports auto-generation of some parts of the model documents (ST 1903 through ST 1908) • Updated example table format to match auto-generated format • Added brief introduction to Motion Imagery Modeling Language (MIML) • Added arrays of Directed Associations

4 Acronyms, Terms, Definitions

IMAP	Integer to floating point MAPping
ISR	Intelligence, Surveillance, and Reconnaissance
KLV	Key Length Value
MIMD	Motion Imagery Metadata
MISB	Motion Imagery Standards Board
MISP	Motion Imagery Standards Profile
ROC_Unknown	Report-on-Change Unknown state
UML	Unified Modeling Language
XML	eXtensible Markup Language

5 Introduction

Motion Imagery metadata facilitates managing, discovering, and providing contextual information for exploiting Motion Imagery. To date the MISB has published numerous metadata standards to address the needs of the ISR Motion Imagery community. New applications demand additions to these standards as well as the creation of new ones, which warrants the MISB to apply a more holistic approach to metadata in general. The purpose of the MIMD suite of standards is to provide a framework that consolidates and organizes information into a singular, unified metadata model, thereby serving many use cases. This MIMD model then transmutes into different formats, such as KLV and XML.

The MIMD standards address different aspects of the MIMD system, which defines the processes for generating efficient KLV (and other future formats) from a UML model. Figure 1 identifies the five components of the MIMD system: MIMD Modeling Rules, MIMD Model, MIMD Instance, MIMD Transmutation Instructions, and MIMD Formats. The blue items represent MISB standards and the green items represent implementation data.

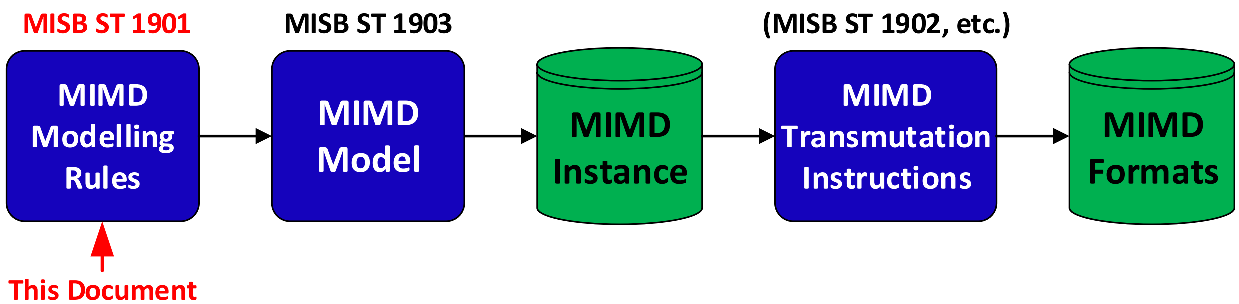

Figure 1: MIMD System Components

The MIMD Modeling Rules comprise a subset of the UML rules for building the MIMD Model. These rules enable transforming, or transmuting, MIMD Instances into MIMD Formats by the MIMD Transmutation Instruction documents.

The MIMD Model is a UML data model following the MIMD Modeling Rules. The Model contains classes, class attributes and defines relationships between classes. MISB ST 1903 and its supporting documents define the MIMD Model.

A MIMD Instance is a data structure implementation of the MIMD Model. While a class defines the allowed contents and rules, an instance is a specific implementation of the class. In other words, a class is the blueprint; the instance is the actual data.

The MIMD KLV Transmutation Instructions provides the “algorithm” for converting a MIMD Model Instance into MIMD Formats. Currently, the only MIMD Format is KLV. MISB ST 1902 defines the MIMD Model-to-KLV Transmutation Instructions that maps instances and their relationships to hierarchal KLV structures (e.g., Local Sets and Packs). Additionally, these instructions define methods for the “special” model functions for KLV-unique encodings (e.g., temporal compression). The MISB intends to develop other transmutation instructions for other formats such as XML and JSON as needed.

The MIMD Formats are the final product of the MIMD system. The MIMD KLV Format is a bit-efficient implementation of the MIMD Instance data. Systems generate custom subsets of the MIMD KLV by selecting appropriate classes and class attributes for their system and following the MIMD Model-to-KLV Transmutation Instructions and requirements. Systems that create metadata and transmute it to MIMD Formats are producers. Systems that decode MIMD Formats and utilize the metadata are receivers.

6 MIMD Model Overview

The MIMD Model is a set of hierarchical UML class diagrams and textual class definitions based on the rules defined in Section 7 and Section 8. Section 7 defines the subset of the OMG

UML class specification the MIMD Model uses. Section 8 defines extensions beyond the OMG specification including information the MIMD model needs for different encodings (e.g., KLV or XML) and clear definitions for classes along with their attributes.

The MIMD Model defines the allowed structure and relationships of classes that systems may use to report their metadata. At run-time, systems create instances of the classes they need from the MIMD Model definition; systems therefore will, in general, use a subset of the full MIMD Model.

Systems report metadata for discrete [system defined] time. The collection of class instances for a given time defines the content of a MIMD Packet. Each MIMD Packet contains a single MIMD class instance which includes hierarchical children class instances. The MIMD class instance is for an instant in time, but individual child class instances may include temporal offsets (either positive or negative) to fine tune the timing for all the instance's attribute measurements. The temporal offset applies to all the child class attribute measures in the same manner. For example, a class instance with two attributes (waterTemp and seaState) and a temporal offset of +7 nanoseconds, means both waterTemp and seaState were measured 7 nanoseconds after the class's timestamp.

A sequence of MIMD Packets forms a MIMD Stream. From packet to packet, systems can expand (add additional) or contract (remove) class instances within a MIMD Stream. Ideally each packet is a complete and independent MIMD Packet. To conserve bandwidth, the Report-on-Change method may be employed (see the Motion Imagery Handbook [4]). With the Report-on-Change method, each MIMD Packet optionally reports only changes from prior packets, thus eliminating redundant information. The Report-on-Change methodology prescribes rules for data expiration and control. The Report-on-Change includes an "Unknown" data state for undefined items and data items which have expired. For clarity, the MIMD documents use the term "ROC_Unknown" for the "Unknown" state name.

Figure 2 illustrates a series of MIMD Packets occurring over time comprising a MIMD Stream. In this example, each packet has a different MIMD instance structure. Each packet in the stream provides metadata for a given time. The colored squares represent different class instances (e.g., red is platform, green is environmental, brown is sensor). In this example, receivers use the Report-on-Change concept to infer the same constant environmental instance (green) in Packets 2, 3, and 4 from Packet 1. Packet 5 shows a new class instance (in blue) un-reported in the stream prior. Systems may also include or exclude attributes within each class instance, as necessary.

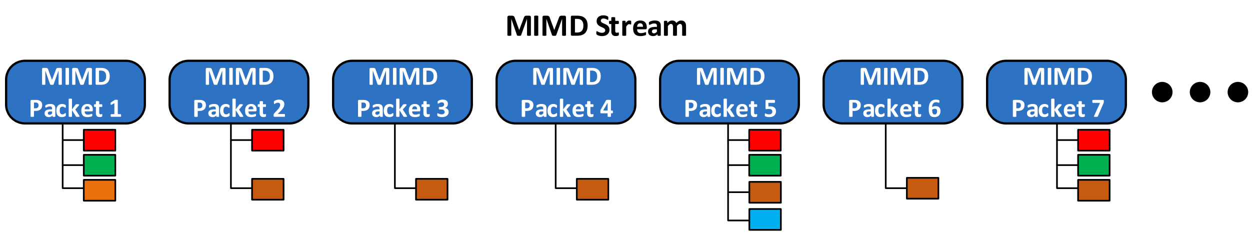

Figure 2: MIMD Stream comprised of MIMD Packets

7 MIMD UML Modeling Rules

The MIMD UML Modeling Rules are a subset of the UML class diagram methodology. The MIMD Model diagram is a hierarchy of **MIMD Model Classes** connected using **MIMD Model Relationships**. The MIMD Model invokes requirements in MISB ST 0107 [5] for Report-on-Change.

7.1 MIMD Model Classes

MIMD Model Classes are UML classes with a name and one or more attributes. A class is a collection of information about a given object (a thing, e.g., Sensor) or topic (a concept, e.g., Counting People). For example, a Sensor class contains information about a sensor, such as its location, the number of rows/columns in its sensor array, etc. The class name reflects the collective meaning of the attributes within the class. The class name is a capitalized single word or phrase without spaces; it uses upper camel case for multiple words (e.g., CamelCase). The naming rules allow the class to map readily and consistently into software.

MIMD Model classes do not include functions. The MIMD Model defines only attributes; the data has no functions or transformations during transmission from the source to destination.

Figure 3 shows an example of a class with the name Platform with three attributes, each with their defined data type.

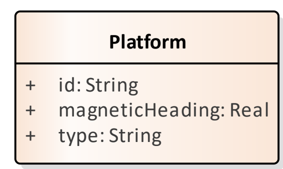

Figure 3: Class Example

7.1.1 Class Attributes

Class attributes are the informational items defining the class content. An attribute's visibility (public, private, etc.) does not apply in the MIMD Model; all attributes are public. Class attributes are not order-specific within the class, and they are in use only one time (i.e., one attribute value per instance per class). Unless otherwise stated, all classes include a set of MIMD Base Attributes and relationships, which are not visible in each classes' diagram. MISB ST 1904 [6] provides details of the MIMD Base Attributes. MIMD UML diagrams require defining two attribute traits: **Name** and **Type**.

The attribute trait **Name** is a single word or phrase starting with a lowercase letter and in lower camel case (e.g., attributeName) for multiple words (i.e., no spaces in the names). The Name may not start with a numeric digit but may include digits after the first letter of the name. Name is unique within the scope of the class, which includes the Base Attributes. Attribute Name equivalence is independent of the case of the name, e.g., "engineTemp" and "engineTEMP" are the same name and not unique. Two *different* classes define two different class-scopes; therefore, each class may have one or more attributes using the same Name as the other class. The naming rules allow the attributes to map easily and consistently into software.

The attribute trait **Type** defines the data representation of the attribute value once the class has been instantiated. There are three categories of Type: *Singular Type*, *Enumeration Type*, and *Array Type.* Additionally, a *Tuple Type* defines a sequence of Unsigned integers (for various MIMD internal uses, e.g., instance's "address" for Directed Association) and a type value of *RESERVED* indicates the potential future use of the attribute identifier defined in Section 8.1.1.

7.1.1.1 Singular Type

The *Singular Type* is the basic data type for defining a single value, for example an integer or decimal number. Ideally these types are logical types for modeling and do not have special values (e.g., NaN) or limitations (e.g., value range, precision) that software languages and physical formats impose. This enables the modeling to be independent of hardware, operating system, and software limitations. The type's value range may (through its attribute definition) specify minimum and maximum traits, which impose constraints. A MIMD Singular Type name is in upper camel case (i.e., the first character must be an uppercase letter). Table 1 lists the allowed MIMD class attribute *Singular Types*. MIMD Transmutation Instructions will specify a type's format for use within the target MIMD Format (e.g., UTF-8 character set from ISO/IEC 10646 [7] for Strings or IEEE 754 floating point value [8] for Reals.)

Table 1: MIMD Class Attribute Singular Types

Type	Definition	Example
String	A sequence of characters supporting multiple language character sets	Model 3
Real	A decimal number (i.e., all rational and irrational numbers)	3.14159
Integer	A signed integer value (e.g., all negative and positive values including zero)	-100
UInt[1]	An unsigned integer value (e.g., all positive values including zero)	127
Boolean	A representation supporting only a true or false values (e.g., 0=false, 1=true)	1

7.1.1.2 Enumeration Type

The *Enumeration Type* is a finite list of enumeration items, one of which assigns to an attribute's value. An enumeration item has three parts: **Enumeration Identifier**, **Name**, and **Definition**. The **Enumeration Identifier** is a unique (within the enumeration list) unsigned integer to identify one of the enumeration items in the list, numerically. The **Name** is a string, unique within the scope of the enumeration list, which identifies one of the enumerations items in the list. The Name does not include spaces but may use underscores ("_") as a word delimiter. The **Definition** describes the meaning of the enumeration or references a document section with further information.

MIMD Model diagrams represent Enumeration Types as an object with a UML stereotype of "≪enumerations≫" and green color shading. One or more classes may link to the enumeration with *undirected* Association. The enumeration object may include the enumeration names in the object, or a supporting document will list the enumeration items. The absence of the names in the enumeration diagram object does not mean it is an empty enumeration; the supporting document will provide the full details. If a class has several uses of the same Enumeration Type, each "line" in the diagram will include an attribute name. If an Array uses an Enumeration Type the "line" in the diagram will include the attribute name and the array notation – see Section 7.1.1.3 for information on Array Types.

[1] UInt renames the UML's UnlimitedNatural data type.

Figure 4 shows a generalized example illustrating a Sensor class referencing an enumeration with the name LensSet.

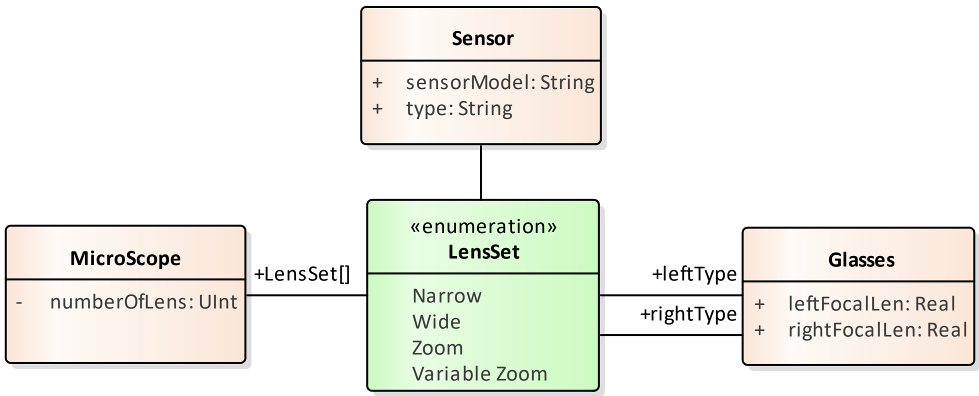

Figure 4: Enumeration Type Example

The Sensor class has only one use of the LensSet Enumeration Type so the attribute name in the Sensor class defaults to LensSet. The MicroScope class has an Array of LensSet Enumeration Types, so the diagram includes the Array Type notation (i.e., LensSet[]); the notation could use a different attribute name. The Glasses class uses the same Enumeration Type twice, so the diagram includes the attribute names for each use.

Table 2 lists the four example enumeration items for the LenSet Enumeration.

Table 2: Example Enumeration LensSet

Identifier	Name	Description
0	Narrow	A *Narrow* lens is a fixed focal length lens zoomed into the scene
1	Wide	A *Wide* lens is a fixed focal length lens zoomed out from the scene
2	Zoom	A *Zoom* lens has several fixed focal length settings each with different “zoom” levels
3	Variable_Zoom	A *Variable Zoom* lens is a continuous zoom lens

7.1.1.3 Array Type

The *Array Type* is a list of Singular Types, Enumeration Types, or References arranged as a multi-dimensional array. Arrays of numerical types (i.e., Real, Integer, and UInt) share the same minimum, maximum, and default precision trait values, if applicable. The *Array Type* notation specifies the type for each Singular Type element within the array followed by multiple dimension declarations. There are three methods for defining dimensions: Defined Length, Maximum Length, and Open Length.

A Defined Length dimension declaration starts with an open square bracket “[“, a count of the number of elements, then a closed square bracket “]” to complete the definition. The element count defines the exact number of elements the attribute must support. For example, “Integer[10]” indicates an array of ten Integer values.

A Maximum Length dimension declaration starts with an open square bracket "[" a less than or equal to sign, "<=", a count of the maximum number of elements, then a closed square bracket "]" to complete the definition. The element count defines the maximum number of elements the attribute supports, data producers define the actual number at run-time. For example, "Integer[<=10]" indicates an array containing a maximum of ten Integer values.

In an Open Length Array, the count of elements is determined at run-time, so the element count is blank. For example, "Integer[]" means the array size is determined at run-time.

Each array may have more than one dimension, for example Integer[10][][3] is an array with three dimensions: the first array dimension has ten elements (defined in the model); the second dimension is determined at run-time; and the third dimension (defined in the model) has three elements. If at run-time the second dimension becomes five, then the array would have a total of 10 * 5 * 3 = 150 elements.

Requirement(s)	
ST 1901-01	Where an Array Type attribute specifies a Defined Length element count for a dimension and the attribute is used by an instance, the instance shall use the specified number of elements for that dimension in the array.
ST 1901.1-06	Where an Array Type attribute specifies a Maximum Length element count for a dimension and the attribute is used by an instance, the instance shall use at most the specified number of elements for that dimension in the array.

7.1.1.4 Tuple Type

The Tuple Type is typically a short sequence of Unsigned Integers (UInts) for use within the MIMD Model's addressing method (for Directed Association) along with other uses. The Tuple Type is the same as a one-dimensional array of UInt, but because of its short length and the restriction to one dimension, it is a separate type to enable more efficient transmutations.

7.1.1.5 RESERVED Type

The RESERVED Type is a placeholder within the attribute list for a planned future attribute. The attribute's Name and Type traits are not known when reserving the attribute; however, the attributes Identifier trait (see Section 8.1.1) is known.

7.2 MIMD Model Relationships

There are three types of MIMD Model relationships: **Composition** ("has-a" dependency relationship), **Inheritance** ("is-a" dependency relationship) and **Directed Association**. The MIMD Model diagrams show all three relationships using standard UML diagram notation.

7.2.1 Composition Relationships

The **Composition** relationship defines a "has-a" connection between a parent and a child class. The parent class "includes" the child class in a similar fashion as a class attribute, i.e., the parent has-a child. The Composition relationship can be singular or include multiplicity (i.e., multiple copies of the child class or a collection of child classes). While a child can define multiplicity, the parent cannot have multiplicities in the MIMD Composition relationship. Section 7.2.1.1 discusses Composition relationship multiplicity.

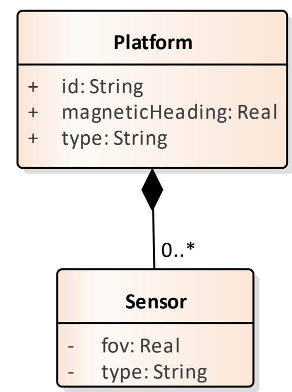

Figure 5: Composition Relationship Example

Figure 5 shows an example UML diagram with two classes (Platform and Sensor) and a Composition relationship between them. The Platform class contains three attributes: id, magneticHeading, and type, each with their associated data types of String, Real, and String, respectively. The Platform class has a fourth attribute -- the Sensor class Composition relationship, shown as a line connecting the two classes together. The filled-in diamond denotes the Platform class is the relationship parent and the Sensor class is the relationship child. Interpret the relationship as "The Platform has-a Sensor." This Composition relationship has a defined multiplicity of 0..*, meaning the Platform has zero or more Sensors (see Section 7.2.1.1). The multiplicity of the Composition relationship means the Platform's attribute of Sensor is a collection of Sensors.

The MIMD Modeling rules do not allow circular (or cyclic) Composition relationship dependency paths in the model; that is, where one class depends on one or more classes that lead back to depending on the first class. For example, a prohibited cycle of three classes is: class A depends on class B, class B depends on class C, and class C depends on class A.

7.2.1.1 Composition Relationship Multiplicity

Composite Relationship Multiplicity enables a parent class to "have-a" collection of child classes. A range of numbers on the composite relationship link defines the multiplicity (e.g., "0..5", or "1..*"). The first number is the required minimum number of elements in the collection and the second number is the required maximum number of elements in the collection. A minimum value of zero indicates the collection is optional; where the minimum value is greater than zero the collection is required when creating an instance of the parent class. A star (i.e., *) for the maximum value indicates the collection does not have an upper bound.

7.2.1.1.1 List Organization

A Composition relationship with multiplicity means the parent class has a collection of child classes.

List organization is important when correlating two lists between two MIMD Packets. For example, in one MIMD Packet a Platform class may have a list of sensors for sensor 1 and sensor 2. In a subsequent packet, receivers need to determine which elements of the list refer to sensor 1 and sensor 2 to correlate with the first packet. The MIMD Model supports two techniques for producers to organize lists: **Fixed Order** and **Marked Element**.

In a **Fixed Order** list, the order of the list remains constant during the entire stream when instantiating the collection of children classes; therefore, every packet uses the same list index to correlate items between packets. Figure 6 illustrates the list correlation in two separate packets when using Fixed Order.

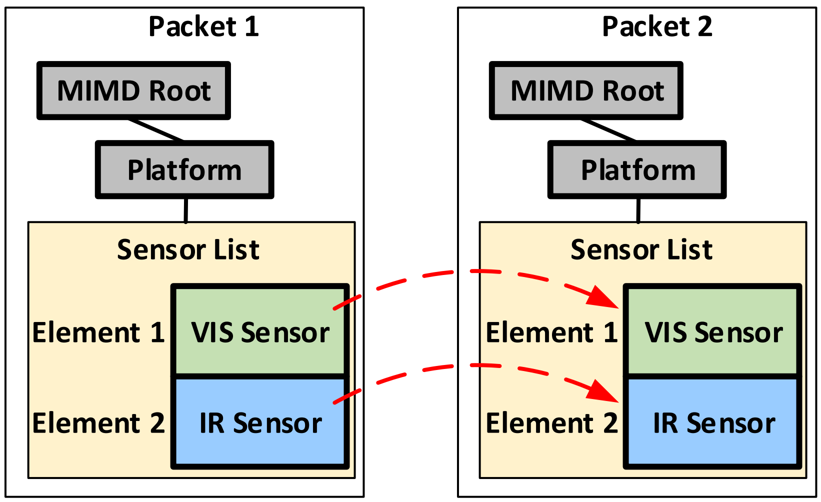

Figure 6: Fixed Order List Element Correlation

In this example the elements correlate using their position, or index, in the list; that is, receivers always know Element 1 will be VIS Sensor and Element 2 will be the IR Sensor.

In a **Marked Element** list each item in the list defines a unique identifier (i.e., MIMD Instance Identifier or mimdId), in every packet; therefore, correlation comes from matching identifiers from packet-to-packet. The mimdId is an attribute within the set of MIMD Base Attributes (see MISB ST 1904) in every MIMD class. Figure 7 illustrates correlating the list for two separate packets when using a Marked Element list. In this example the elements correlate using the mimdId; that is, receivers always know mimdId=13 is for the VIS Sensor and mimdId=17 is for the IR Sensor. This example shows element reversal in the second packet to demonstrate order of elements is irrelevant when using a Marked Element list.

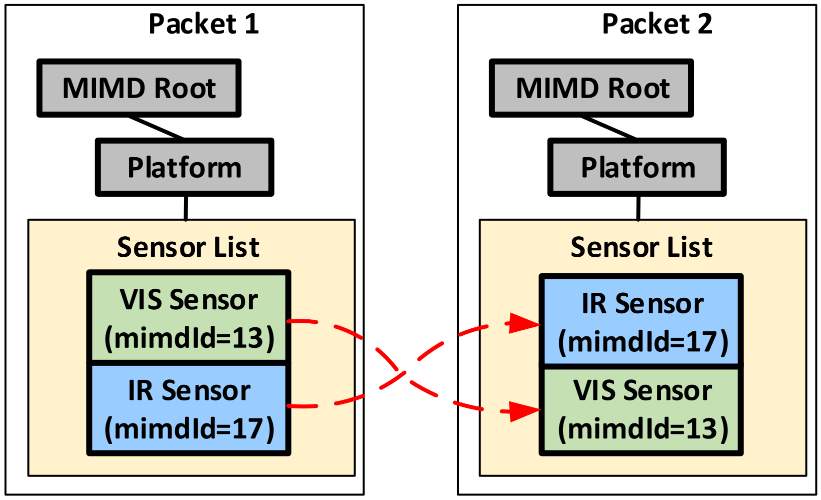

Figure 7: Marked Element List Correlation

The advantage of a Fixed Order over a Marked Element list is fewer bytes and thus less bandwidth by not requiring the mimdId; however, there is a disadvantage when using and modifying a Fixed Order list. Since order is important, a Fixed Order list needs to include all elements up to a changed element. For example, if a list has ten elements (e.g., 10 sensors) and only element seven changes (e.g., sensor 7 becomes inactive), then all elements preceding element seven (e.g., sensors 1-6) need to supply filler elements as placeholders in the list and all elements after seven are optional. The MIMD Transmutation Instructions (e.g., MISB ST 1902) defines the method of providing filler elements.

Lists have operations for: adding elements, inserting elements, and deleting elements. With Fixed Order, new elements may add only to the end of a list. Producers do not arbitrarily insert or delete elements in Fixed Order lists.

The advantage of a Marked Element list is random element order and the reporting of only changing elements (i.e., no placeholders). Producers can add new elements and insert elements anywhere in the list.

When initiating a list, a producer determines whether to use either a Fixed Order or Marked Element list for the duration of the MIMD Stream. Receivers will interpret the list in one of three states: Unknown, Fixed Order, or Marked Element. A list is Unknown until positive confirmation of either Fixed Order or Marked Element is determined. If any element in the list is missing a mimdId, the list is assumed Fixed Order.

Requirement	
ST 1901-02	Where a producer uses a Marked Element List, each element of the list shall include its MIMD Identifier (mimdId).

7.2.2 Inheritance Relationship

The **Inheritance** relationship defines an "is-a" connection between a parent and a child class. The child inherits all the attributes of the parent, so the child "is-a" new extension of the parent class. In effect, the child class is a more specific version of the more general parent class. The MIMD Model does not allow Redefinition (as described in the UML specification) of the MIMD Base Attributes. A child class does not define its own MIMD Base Attributes, instead the child's Base Attributes are the same as the parent's Base Attributes. Neither the child nor parent define multiplicities with an Inheritance relationship.

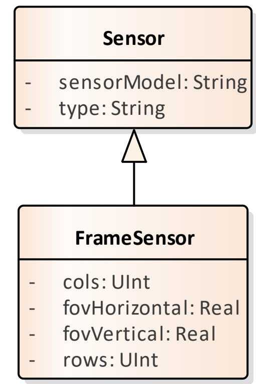

Figure 8: Inheritance Relationship Example

Figure 8 shows an example of an Inheritance relationship. The Sensor class contains two attributes, the sensorModel and type. The Sensor class is a generalization or superclass of the FrameSensor class, where the FrameSensor class includes, or inherits, all attributes of the Sensor class. A FrameSensor has six attributes: cols, fovHorizontal, fovVertical, and rows – plus the Sensor class attributes – sensorModel and type. Interpret the relationship as "FrameSensor is-a Sensor."

The MIMD Modeling rules do not allow circular (or cyclic) Inheritance relationship dependency paths in the model. The MIMD Modeling rules do not allow multiple inheritances; therefore, child classes have Inheritance Relationships with only one parent.

7.2.3 Directed Association Relationship (Class to Class Reference)

The MIMD Model uses **Directed Association** relationships to define a "refers-to" link between a referring instance and destination instance. The referring instance has an attribute that "points" to the destination instance using the destination's unique instance identifier (i.e., mimdId), which is part of the MIMD Base Attributes. The Directed Association relationship does not provide any data containment, such as Composition or Inheritance; it provides only a linkage between class instances.

Figure 9 shows an example of Directed Association from the Encoder class (referring class) to the Sensor class (destination class). The diagram shows a Platform with composition for Sensor(s), and Encoder(s). The Sensor class includes the MIMD Base Attributes, one of which is a unique instance identifier - mimdId (see Section 7.2.3.1). The Encoder includes an attribute, called "source" that contains the Sensor instance's mimdId value. The Base Attributes are not visible in the MIMD class diagrams.

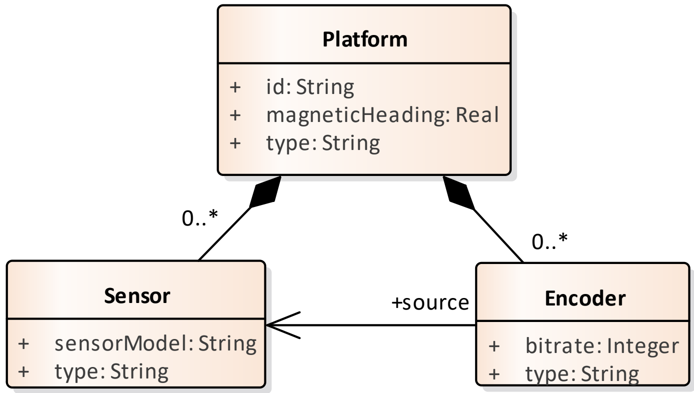

Figure 9: Example classes using Directed Association

During normal system operation, a Sensor passes data to one or more Encoders. Which Sensor connects to which Encoder can change at any time. Within this example the Encoder class is not a composition of the Sensor class; thus, the Directed Association shows which Encoder connects to which Sensor. By having each Sensor class instance define a mimdId, and each Encoder class includes a reference attribute (i.e., source), an Encoder can "link" to a Sensor from which it receives input.

Figure 10 illustrates two pairs of Sensor and Encoder class instances and shows the references for each encoder's source input in red. Each Sensor class instance defines a mimdId, shown in parenthesis, and each Encoder class instance shows what the *source* attribute links to. In the figure, Encoder 1 receives its input from Sensor 1 and Encoder 2 receives its input from Sensor 2. At run-time, if a system needs to change which encoder links to which sensor, the value of *source* changes appropriately.

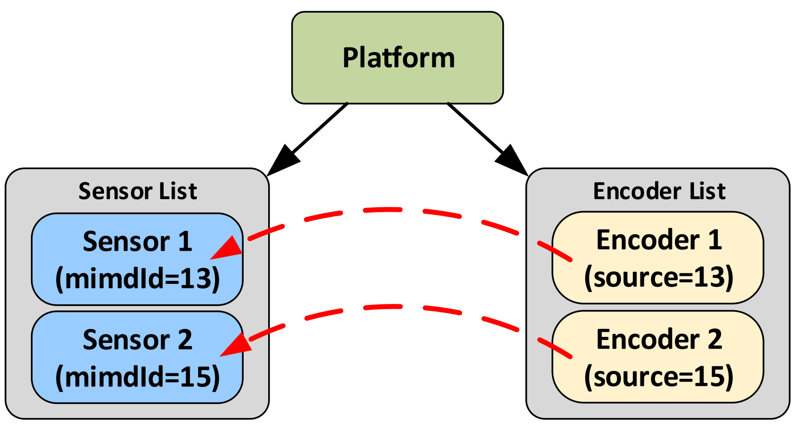

Figure 10: Example Soft Links Between Instances

7.2.3.1 Instance Identifier – mimdId

The mimdId is the class instance's unique identifier, which is a two value Tuple: *serial number* and *identifier group*. The instance serial number is a unique unsigned integer value assigned at run time by a data creator. The identifier group defines a group to which an instance identifier belongs. Systems generating the data assign group identifiers, as needed, to sub-components of the system. Group zero is the default group if the group identifier is not included in the Tuple (some transmutations may remove default group identifiers to save bandwidth).

The instance serial number is unique within the scope of the group so a system using multiple group identifiers may have instances with the same serial number but different group numbers. The combination of instance serial numbers with the group identifier creates a globally unique identifier for every instance in the MIMD model.

MISB ST 1904 provides details on the use of the mimdId within the MIMD model.

7.2.3.2 Arrays of Directed Association Relationships

The MIMD Model allows arrays of Directed Association relationships. This enables a class to define multiple references to the same class multiple times, thereby having a list of instance references. Figure 11 illustrates a Sensor class with a PointOfInterest class list and a PhysicalObject class which references a subset of the points of interest using an array of directed associations.

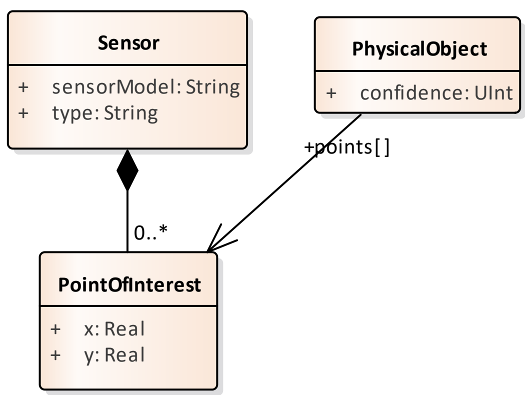

Figure 11: Example Directed Association Array (Reference Array)

7.3 MIMD Model Diagram External References

The complete MIMD Model diagram is large, so a collection of documents (i.e., MISB ST 1903, ST 1904, etc.) divides the diagram into logical groups and provides the details. When a document needs to refer to a class in a different group (e.g., within the current document or another document) the class will have the name of the document, two colons ("::") and the name of the class; furthermore the class will have a yellow highlight. Figure 12 illustrates an example of an external reference where the Platform class refers to the Stage class. This example shows the Stage class in yellow. All the Stage class's children and references are not visible; however, as indicated with the "ST1906" designation, the Stage class's full diagram, attributes, and details (including its children and references) are available in MISB ST 1906.

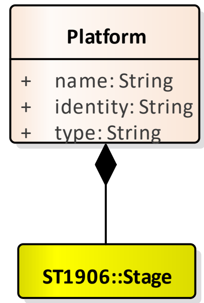

Figure 12: External Reference Example

8 MIMD Textual Class Definitions

The MIMD UML Modeling Rules in Section 7 define a graphical depiction of the MIMD Model. The MIMD Textual Class Definitions augment the graphical model, defining a written description of the MIMD Model. The textual model enables transitioning the UML model into software and aids in encoding the MIMD Model with different formats (e.g., KLV or XML). Each MIMD standard defining a MIMD class includes a table definition for the class. The table provides relationship and attribute details including all the traits and a full description of each attribute's purpose. Section 8.4 describes the table columns and provides an example of a graphical class model and its table definition.

The MIMD Textual Class Definitions provide the rules and requirements for detailing information about each class and its attributes. These rules and requirements build upon the MIMD UML Modeling Rules in Section 7. The textual rules and requirements include additional attribute traits and defines class relationships and enumerations as attributes with special types.

8.1 Additional Attribute Traits

Section 7.1.1 defines the basic attribute traits (i.e., Name and Type). The MIMD Textual Class Definitions define an additional required trait (i.e., Identifier) and other optional traits depending on the type of attribute.

8.1.1 Identifier Trait

The attribute trait **Identifier** is a required trait; it is an unsigned integer unique within the scope of the class. The MISB assigns permanent attribute Identifiers which ensures backwards compatibility with future updates to the model. Since an Identifier is unique only within the class scope, two different classes may have an attribute with the same Identifier. For example, a Platform class has a magnetic heading (magneticHeading) attribute with an Identifier equal to 33, while a Sensor class has a field-of-view (fov) attribute with an Identifier equal to 33. However, within the *scope* of the class the Identifier of 33 is unique across all other Identifiers. During KLV Transmutation (see MISB ST 1902) the attribute Identifier becomes a Local Set tag.

8.1.2 Optional Traits

The optional traits are: **Minimum**, **Maximum**, **Default Resolution**, **Maximum Length**, **Units**, and **Deprecate**. Optional traits apply to the attribute under certain conditions; for example, with an attribute type Real the Minimum trait may apply. Using all three numeric traits (Minimum, Maximum, and Default Resolution) on a Real attribute signals the use of the KLV IMAP (MISB ST 1201 [9]) algorithm – see MISB ST 1902.

8.1.2.1 Minimum and Maximum

The attribute traits **Minimum** and **Maximum** provide the numeric bounds for an attribute of numeric type (Real, Integer, or UInt) and define the limits of Composite Relationship multiplicities. For numeric types both the Minimum and Maximum traits are values with the Minimum value less than the Maximum value. These two traits enable range checking and provide information for the KLV IMAP algorithm when encoding floating point types. The Minimum and Maximum trait values may be stated as numeric values or use the constants in Table 6 of Appendix A. The Minimum and Maximum trait numeric values follow the International System of Units (SI Units) [10] method for “Formatting numbers, and the decimal marker” by using a space for digit grouping and a period for the decimal marker.

For Composite Relationship multiplicities the Minimum and Maximum trait values define the required minimum and maximum number of elements in the collection.

Requirement	
ST 1901-03	Where a numeric attribute specifies the Minimum and Maximum traits, the instance value shall not exceed the numeric range defined by these bounds.

8.1.2.2 Default Resolution

The attribute trait **Default Resolution** provides a resolution value for a Real quantity. Producers may override the default value at runtime to meet application requirements. The KLV IMAP (MISB ST 1201) algorithm specifies the use of a resolution (the default or overriding resolution), along with the minimum and maximum values to encode floating point values. The Default Resolution numeric value follows the International System of Units (SI Units) method for “Formatting numbers, and the decimal marker” by using a space for digit grouping and a period for the decimal marker. The Default Resolution trait value may be stated as numeric values or use the constants in Table 6 of Appendix A.

Integer and UInt types have a fixed resolution of one (1). All Integer and UInt attributes do not define the resolution trait.

8.1.2.3 Maximum Length

The attribute trait **Maximum Length** states the maximum number of elements a value can contain. For example, a String value may have a limited length imposed by the model. The Maximum Length applies to the String and Tuple types. For strings the Maximum Length is the maximum number of characters (e.g., UTF-8 characters). For Tuples, the Maximum Length is the number of UInts. The Maximum Length is **<u>not</u>** a maximum number of bytes.

Requirement	
ST 1901-04	Where an attribute specifies the Maximum Length trait, the instance value shall be limited to the number of elements (e.g., characters) defined by the Maximum Length.

8.1.2.4 Units

The attribute trait **Units** provides the unit of measurement for the quantity indicated. Models will specify all Units in International System of Units (SI Units). Model developers are prohibited from creating attributes with the same meaning as another attribute but with different units or measurement scales.

8.1.2.5 Deprecate

The attribute trait **Deprecate** denotes the attribute is invalid for use (i.e., deprecated). Section 10 details how and when to apply this trait.

8.2 Relational and Enumeration Attributes

The MIMD UML diagrams list relational and enumeration attributes as lines (e.g., composite relationships) between classes. The MIMD Textual Class Definitions are a written (non-graphical) specification of the model so a class includes the relationships and enumeration types as attributes. The following sections define the textual representation of the three types of MIMD relationships (from Section 7.2) and the Enumeration Type (from Section 7.1.1.2); that is, Composite Relationship, Inheritance Relationship, Directed Association, and Enumeration Type.

8.2.1 Composite Relationship

The Composite Relationship is a “has-a” relationship where a parent class “has-a” child class. The MIMD system treats this relationship as containment, where the parent contains the child classes attributes within a single attribute of the parent classes attributes. Composition relationships form the model hierarchy and may refer to a single class or a collection of the same class. The Composite Relationship defines containment of the child class within the context of the parent class, so when encoding the child class’s information (in any format such as KLV or XML) the child’s attribute data is embedded within the parent’s class data in some form which is decodable back into a separate child class. Section 7.2.1 discusses the graphical model for Composition Relationships.

The parent class of a Composite Relationship textually represents a child class as another attribute in the parent class. Each singular Composition relationship defines the Name, Type, Identifier, and optionally Deprecate traits. By default, the Name trait is set to the child class’s name but with the first letter in lowercase to comply with the attribute naming rules. The Type trait for a singular Composition relationship is also the child class’s name. The Type trait for a Composition relationship with multiplicities is “LIST <*x*>”, where *x* is the child class’s name. The Identifier trait is an unsigned integer unique within the scope of the parent class’s attribute list. If a parent class needs two or more Composition relationships to the same child class, each relationship will have unique Name and Identifier traits. The Minimum and Maximum trait values define the number elements in a Composite Relationship multiplicity. Where there is no Composite Relationship multiplicity the Minimum and Maximum traits are set to "--" (double dash).

Section 8.4 includes an example of a composite relationship in the class table. The row with Id 41 is an example of a single composite relationship attribute, and row 42 is an example of a Composite Relationship attribute with multiplicities.

8.2.2 Inheritance Relationship

The Inheritance Relationship is a merging of parent and child attributes into a single "combined" class. Ideally, the merging would form a new class with all the attributes from the parent and child in one class; however, merging attribute traits (specifically the Name and Identifier traits) causes a modelling data-management issue. To rectify the data-management issue the parent class contains the child class the same way as the Composite Relationship does, i.e., as an attribute.

A parent class may have multiple inheriting children; however, during implementation, inheritance rules allow only one inheritance relationship per parent when instantiating a class. For example, a Position class may have an inheriting child class for Geodetic position and another inheriting child class for Geocentric position. When instantiating a Geodetic [and therefore the Position] class, the Position class only contains the Geodetic attribute; the UML inheritance rules forbid including the Geocentric attribute in the Position class. Section 7.2.2 discusses the graphical model for inheritance relationships.

Each Inheritance relationship defines an attribute with Name, Type, Identifier, and optionally Deprecate traits. By default, the Name trait is set to the child class's name but with the first letter in lowercase to comply with the attribute naming rules. The Type trait is the child class name. The Identifier trait is a unique number within the scope of the parent classes attribute list. The Identifiers for the child class attributes are independent of the parent class, so they start at the value of 33.

Section 8.4 includes an example of an inheritance relationship in the class table. The row with Id 46 is an example of an inheritance relationship attribute.

8.2.3 Directed Association (Class to Class Reference)

The Directed Association is a reference between a referring instance and destination instance within the model. The reference only identifies an instance in the hierarchy, i.e., it is a "pointer" to an instance. The referring class textually represents the Directed Association as an attribute in its class. Unlike the Composite Relationship Type, the destination instance is not embedded within the referring class; only the address (i.e., mimdId) of the destination instance is included in the referring class. Section 7.2.3 discusses the Directed Associations.

Each Directed Association relationship defines the Name, Type, Identifier, and optionally Deprecate traits. The Name trait is a unique name within the classes attribute list. The Type trait is "REF<x>", where x is the destination class's name, e.g., the type of a reference to a "Sensor" class is "REF<Sensor>". The Identifier trait is a unique number within the scope of the parent classes attribute list.

Section 8.4 includes an example of a directed association in the class table. The row with Id 44 is an example of a directed association attribute.

Arrays of Directed Association use the bracket notation after the “REF<x>” to denote an array of references, e.g., “REF<Sensor>[]” indicates an array of references to Sensor class instances.

8.2.4 Enumeration Type

An Enumeration Type is a list of Enumeration Items one of which assigns to an attribute’s value. The class using an Enumeration Type textually represents the enumeration use as another attribute in the class. Section 7.1.1.2 defines the details of each Enumeration Item.

Each class which refers to an Enumeration Type defines an attribute with Name, Type, Identifier, and optionally Deprecate traits. By default, the Name trait is set to the Enumeration Type’s name but with the first letter in lowercase to comply with the attribute naming rules. If a single class uses the same Enumeration Type multiple times, each attribute in the class will have a unique name and identifier. The Type trait is the Enumeration Type’s name. The Identifier trait is a unique number within the scope of the parent classes attribute list.

Section 8.4 includes an example of an Enumeration Type in the class table. The row with Id 43 is an example of an Enumeration Type attribute.

8.3 MIMD Model Document Structure

MIMD Model documents follow the similar structure of all MISB document format. MIMD Model documents have two required top-level sections:

- **Model Classes** – Serial list of all classes, with their attributes, and one or more UML diagrams visually representing the classes. Each class defines an Attribute Table; Section 8.4 defines the common table structure.
- **Model Enumerations** – List of all enumeration objects, with their list of individual items including the identifier, name, and a brief description. Section 8.5 provides an example of the Enumeration section.

8.4 Attribute Table Format

All class definitions will include an attribute table which lists all the class’s attributes (except the base attributes), and their traits. The table has nine columns of trait and descriptive information: **Id**, **Name**, **Type**, **Min**, **Max**, **Res**, **MLen**, **Units**, and **Ref**.

- The **Id** trait is the unique numerical identifier for the attribute; that is, unique within the scope of the class. The Id is an unsigned integer starting at a value of 33 for all class descriptions (the Base Attributes reserve Id’s 1 through 32).
- The **Name** trait is a unique name (within the scope of the class) for the attribute. Section 7.1.1 defines the rules for the attribute Names.
- The **Type** trait defines the data representation of the attribute value once the class has been instantiated. There are four categories of Types: Singular, Array, Enumeration, and Relationship.

- Section 7.1.1.1, Section 7.1.1.2, and Section 7.1.1.3 define Singular, Enumeration Types and Array, respectively.
- MIMD Models use three kinds of Relationships: Composite, Inheritance, and Directed Association.
    - The Type names for Composite are the child class name. The Type names for Composite Relationship lists prepend the child class name with “LIST<” and have a suffix of “>”, for example “LIST<Sensor>”.
    - The Type name for Inheritance is the name of the child class.
    - The Type names for Directed Association prepend the child class name with “REF<” and have a suffix of “>”, for example “REF<Sensor>”.

- The **Min**, **Max**, and **Res** traits are for numerical Types and Composite Relationship multiplicity bounds; otherwise, they are set to either “N/A” for Not Applicable, or “--" (double dash) for unknown. For numeric types, the Min and Max are the Minimum and Maximum bounds of the numeric attribute. For Composite Relationship multiplicities, the Min and Max are the allowed number of child elements in the multiplicity collection. A min value of zero indicates the whole list may be empty (and therefore optional). A min value of greater than zero indicates the minimum required number of elements in the list. A “*” for the max represents an unlimited maximum number of elements.
- The **MLen** trait defines the Maximum Length for a String or Tuple Type; otherwise, it is set to “N/A” for Not Applicable.
- The **Units** trait defines the units of measure for the attribute or “None” if units do not apply. All units follow the SI standard.
- The **Ref** column provides a sub-section reference that is one level below the class section level (e.g., if a class’s section number is 7.2, the class attributes sub-sections’ will start at 7.2.1.

Table 3 summarizes the abbreviations found in a Class Attribute Table.

Table 3: Class Attribute Abbreviations

Word or Phrase	Abbreviation
Identifier	Id
Name	Name
Type	Type
Minimum	Min
Maximum	Max
Resolution	Res
Maximum Length	MLen
Units	Unit
Reference	Ref
Deprecate	Deprecate
Not Applicable	N/A
Unknown	--

Figure 13 shows a hypothetical class called ExampleClass to illustrate the diverse types of attributes and relationships. The diagram shows other classes also but without their attributes.

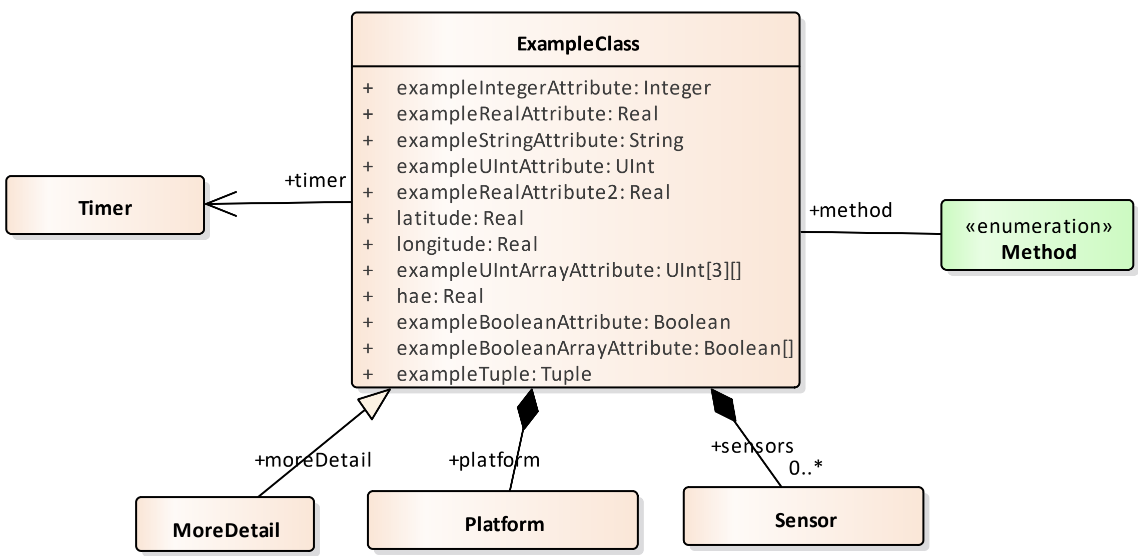

Figure 13: Example Model for Attribute Table Example

Table 4 shows an attribute table for the ExampleClass class in Figure 13.

Table 4: Example Class Attribute Table

Id	Name	Type	Min	Max	Res	MLen	Units	Ref
33	exampleIntegerAttribute	Integer	-100	128	N/A	N/A	ns	8.4.1
34	exampleRealAttribute	Real	--	--	.01	N/A	m	8.4.2
35	exampleStringAttribute	String	N/A	N/A	N/A	120	None	8.4.3
36	exampleUIntAttribute	UInt	0	53	N/A	N/A	None	8.4.4
37	exampleRealAttribute2	Real	0	PI	1E-6	N/A	rad	8.4.5
38	latitude	Real	-PI	PI	1E-6	N/A	rad	8.4.6
39	longitude	Real	-PI	PI	1E-6	N/A	rad	8.4.7
40	exampleUIntArrayAttribute	UInt[3][]	--	--	N/A	N/A	None	8.4.8
41	platform	Platform	N/A	N/A	N/A	N/A	None	8.4.9
42	sensors	LIST<Sensor>	0	*	N/A	N/A	None	8.4.10
43	method	Method	N/A	N/A	N/A	N/A	None	8.4.11
44	timer	REF<Timer>	N/A	N/A	N/A	N/A	None	8.4.12
45	hae	Real	-900	20000	0.5	N/A	m	8.4.13
46	moreDetail	MoreDetail	N/A	N/A	N/A	N/A	None	8.4.14
47	exampleBooleanAttribute	Boolean	N/A	N/A	N/A	N/A	None	8.4.15
48	exampleBooleanArrayAttribute	Boolean[]	N/A	N/A	N/A	N/A	None	8.4.16
49	exampleTuple	Tuple	N/A	N/A	N/A	2	None	8.4.17

The following sub-sections provide example descriptions for the attributes in Table 4. Normally the descriptions discuss the meaning of the attribute and do not repeat information from the attribute table as these do.

8.4.1 Attribute 33 - exampleIntegerAttribute

Attribute 33 is an Integer attribute with a minimum allowed value of -100 and maximum allowed value of 128. Because the attribute is an Integer the resolution is one nanosecond.

8.4.2 Attribute 34 - exampleRealAttribute

Attribute 34 is a Real attribute with an unlimited minimum and maximum value. The resolution is one hundredth of a meter (or one centimeter).

8.4.3 Attribute 35 - exampleStringAttribute

Attribute 35 is a String attribute with a maximum of 120 characters (e.g., UTF-8).

8.4.4 Attribute 36 - exampleUIntAttribute

Attribute 36 is an unsigned integer (UInt) attribute with a minimum allowed value of zero (by definition of UInt) and a maximum allowed value of 53. The resolution is one and there are no units for this value.

8.4.5 Attribute 37 - exampleRealAttribute2

Attribute 37 is a Real attribute with a minimum allowed value of -PI (i.e., $-\pi$) and a maximum allowed value of PI (i.e., π), which is a constant defined in Appendix A. The resolution is 1E-6, or 1×10^{-6} radians.

8.4.6 Attribute 38 - latitude

This section demonstrates the use of a combined description of attributes 38, 39, and 45 (attribute sections for attributes 39 and 45 “refer” to this section). The latitude, longitude, and hae attributes are three parts of a coordinate so they share a combined description. Shared descriptions do not need to be sequential attributes in the attribute table, as shown with hae, Id=45.

8.4.7 Attribute 39 – longitude

Refer to attribute 38, latitude, for the details of this attribute.

8.4.8 Attribute 40 - exampleUIntArrayAttribute

Attribute 40 is an example of a two-dimensional array of Unsigned Integers, where the number of rows is 3 and the number of columns is determined at runtime.

8.4.9 Attribute 41 - platform

Attribute 41 is an example of a composite relationship attribute.

8.4.10 Attribute 42 - sensor

Attribute 42 is an example of an attribute that is a collection of child classes. The collection is optional since the minimum trait is zero. The collection may include an unlimited number of elements since the maximum trait value is set to star (*).

8.4.11 Attribute 43 - method

Attribute 43 is an example of an Enumeration, where Table 5 defines the *Method* Enumeration

8.4.12 Attribute 44 - timer

Attribute 44 is an example of a reference to a Timer class.

8.4.13 Attribute 45 – hae

Refer to attribute 38, latitude, for the details of this attribute.

8.4.14 Attribute 46 - moreDetail

Attribute 46 is an example of an inheritance relationship to the ExampleClass.

8.4.15 Attribute 47 - exampleBooleanAttribute

Attribute 47 is an example of a single Boolean attribute.

8.4.16 Attribute 48 - exampleBooleanArrayAttribute

Attribute 48 is an example of an array of Boolean values.

8.4.17 Attribute 49 – exampleTuple

Attribute 49 is an example of a "two" Tuple which supports two UInts.

8.5 Enumeration Section Format

Enumeration tables consist of three columns: Identifier, Name, and Description. Section 8.5.1 illustrates an enumeration section.

8.5.1 Method Enumeration

This shows an example enumeration section. See Table 5 for list of enumeration values.

Table 5: Enumeration Values for Method

Identifier	Name	Description
0	Fire	Target is burning
1	Water	Target is wet
2	Air	Target is flying

9 Motion Imagery Modeling Language

The Motion Imagery Modeling Language (MIML) is a concise textual representation of a model following the rules from Section 7 and Section 8. The model includes: the class definitions with their attributes plus their traits, the relationships between classes, and enumerations. A formal grammar which enforces the rules and structure of the model defines the syntax of MIML. Future updates of the MIMD model standards will use the MIML description as the normative form of the MIMD model.

10 Model Management

To reduce backwards compatibility issues for MIMD Model developments and revisions, all updates to attributes and classes will be additive. Model developers have three possible options for changes to class attributes and classes: Create, Delete, and Update.

1. Create - When creating a new attribute, it will have a unique Name and Identifier within the scope of its class. Similarly, all new classes will have unique names.
2. Delete - To delete an attribute, the attribute's Deprecate trait is set to the current version, meaning the attribute is not for use in any application from the named version forward.

Model developers are not to reuse the attribute's Name or Identifier for any other attribute in the class. When deprecating all relationships to a class, including association attributes via the mimdId, the whole class is no longer a part of the model – i.e., it is deleted.

3. Update - Model developers will not make updates to existing attributes that change any of the attribute's traits. Instead, if an attribute needs an update, model developers *create* (see above) a new attribute and *delete* (see above) the existing attribute. Model developers will not change class names but instead create new classes and the appropriate relationship attributes.

There are two version incompatibility situations: (1) old version producer to new version receiver and (2) new version producer to old version receiver. For case (1) if a new version receiver only accepts the new version model data, the receiver will ignore all attributes marked for deprecation in the new model. Attributes common in the old and new versions will function normally.

For case (2) an old version receiver must ignore attributes it does not know about and expect some attributes to be missing (i.e., if the new version had them deleted). All attributes common in the old and new version will function normally.

Requirement	
ST 1901-05	When a receiver detects an unknown attribute, the receiver shall ignore or skip the attribute.

Appendix A Constants

Attribute traits may use the constants listed in Table 6.. To clearly indicate a value is a constant in the attribute tables, all constant names are CAPITALIZED, individual words are separated by underscores (e.g., CONSTANT_NAME. For irrational numbers, the values in this table only specifies up to 15 digits to provide enough precision for double precision IEEE values.

Table 6: MIMD Modelling Constants

Constant	Value	Description
PI	3.141592653589793	The circumference of a circle divided by its diameter. The value of PI (or π) is irrational and has an infinite number of digits after the decimal point. In Radians a value of PI equals 180 degrees, or a half circle.
HALF_PI	1.5707963267948965	One half the value of PI, i.e., $\frac{\pi}{2}$. In Radians the value of HALF_PI equals 90 degrees, or a quarter of a circle.
TWO_PI	6.283185307179586	Two times the value of PI, i.e., 2π. In Radians the value of TWO_PI equals 360 degrees, a full circle.